

# TRABALHO PRÁTICO 2 - protocol HTTPS over TLS



**Universidade de Brasília**

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA DE SEGURANÇA DA COMPUTACIONAL- 2024.2

## TRABALHO PRÁTICO 2 - protocol HTTPS over TLS

MEMBROS:

VINÍCIUS BOWEN - 180079239

RAMON OLIVEIRA - 242039630

---

### Abstract: Implementação do Servidor e Cliente HTTPS

A segurança na comunicação entre cliente e servidor é um dos principais desafios no desenvolvimento de aplicações web. O protocolo HTTPS (HyperText Transfer Protocol Secure) surge como uma solução garantindo confidencialidade, integridade e autenticidade dos dados transmitidos. Para compreender melhor seu funcionamento, este relatório apresenta a implementação de um servidor e cliente HTTPS utilizando a linguagem Python, explorando bibliotecas como `http.server`, `ssl` e `cryptography`. Além da implementação prática, são descritas estratégias de validação e testes da comunicação segura, utilizando ferramentas como OpenSSL e Wireshark.

## 1. Introdução: Pesquisa dos Protocolos de Segurança

## 1.1. Detalhamento do HTTPS, SSL e TLS

### SSL (Secure Sockets Layer) e TLS (Transport Layer Security)

O SSL e seu sucessor, TLS, são protocolos essenciais para garantir a segurança das comunicações na internet. Eles utilizam criptografia para proteger dados transmitidos entre clientes e servidores, impedindo que terceiros acessem informações sensíveis como senhas e dados financeiros. O TLS surgiu como sua evolução, aprimorando a segurança e corrigindo vulnerabilidades.

Os principais objetivos desses protocolos são:

-

**Confidencialidade:** Utilizam criptografia simétrica e assimétrica para evitar que os dados sejam interceptados.

-

**Autenticação:** Certificados digitais asseguram que o servidor (e, em alguns casos, o cliente) é legítimo.

-

**Integridade dos Dados:** Algoritmos de hash garantem que os dados não foram alterados durante a transmissão.

### Etapas de Segurança e Principais Algoritmos

1. **Criptografia:** Usa algoritmos como AES (Advanced Encryption Standard), ChaCha20 e 3DES para criptografar os dados transmitidos.
2. **Autenticação:** Certificados digitais X.509 são utilizados para autenticar a identidade do servidor e, opcionalmente, do cliente.
3. **Troca de Chaves:** Utiliza o algoritmo Diffie-Hellman, ECDHE (Elliptic Curve Diffie-Hellman Ephemeral) ou RSA para estabelecer uma chave de sessão segura. Essa troca de chave pode sofrer um "padding" utilizando uma técnica como o OEAP.
4. **Integridade dos Dados:** Funções hash como SHA-256 garantem que os dados não foram modificados durante a transmissão.

### Resumo das Versões e Evolução

- **SSL 1.0:** Nunca foi lançado publicamente devido a falhas de segurança.
- **SSL 2.0:** Lançado em 1995, mas rapidamente substituído devido a vulnerabilidades.

- **SSL 3.0:** Lançado em 1996, trouxe melhorias, mas foi considerado inseguro com o tempo.
- **TLS 1.0** (1999): Primeira versão do TLS, substituiu o SSL 3.0.
- **TLS 1.1** (2006): Melhorou a segurança contra ataques de injeção de pacotes.
- **TLS 1.2** (2008): Introduziu novos algoritmos de criptografia e hash mais seguros.
- **TLS 1.3** (2018): Tornou o handshake mais eficiente e removeu algoritmos inseguros.

## 1.2. HTTPS (HyperText Transfer Protocol Secure)

O HTTPS é a versão segura do HTTP, utilizando SSL ou TLS para criptografar a comunicação entre o navegador e o servidor. Esse protocolo garante que os dados transmitidos sejam confidenciais e íntegros, protegendo contra ataques de interceptação e manipulação, como os ataques Man in the Middle.

Atualmente, mais de 80% dos sites na web utilizam TLS para garantir a segurança das comunicações. O protocolo HTTPS utiliza algoritmos de criptografia robustos, como RSA, para assegurar que apenas o destinatário pretendido possa ler a mensagem, mantendo a privacidade e a integridade dos dados transmitidos.

O HTTPS evoluiu junto com os protocolos SSL e TLS, adotando as melhorias de cada versão para oferecer mais segurança e desempenho.

---

# 2. Implementação do Servidor e Cliente HTTPS

## 2.1. Visão Geral da Arquitetura Cliente/Servidor

A comunicação segura entre um cliente e um servidor HTTPS envolve a troca de informações criptografadas utilizando o protocolo TLS. O fluxo básico dessa comunicação é:

1. O cliente estabelece conexão com o servidor via HTTPS.
2. O servidor apresenta seu certificado digital (SSL/TLS).

3. O cliente verifica a autenticidade do certificado.
4. O cliente e o servidor realizam um handshake TLS para troca de chaves seguras.
5. A comunicação entre cliente e servidor ocorre de forma criptografada.

## 2.2. Utilização do OpenSSL e Outras Bibliotecas de Segurança

Para implementar um servidor e cliente HTTPS, utilizaremos **Python** com as bibliotecas:

- `http.server` e `ssl` para criar o servidor HTTPS.
- `requests` e `socket` para implementar o cliente HTTPS.
- `cryptography` para geração de certificados SSL/TLS via código.
- `datetime`: Usado para definir o período de validade do certificado.

## 2.3. Implementação Prática e Explicação do Código

A seguir, mostramos a implementação do servidor HTTPS em Python:

```
import http.server
import ssl
from cryptography import x509
from cryptography.hazmat.primitives import hashes, serialization
from cryptography.hazmat.primitives.asymmetric import rsa
import datetime

# Gera uma chave privada RSA de 2048 bits
key = rsa.generate_private_key(public_exponent=65537, key_size=2048)

# Cria um certificado autoassinado válido por 1 ano
subject = issuer = x509.Name([
    x509.NameAttribute(x509.NameOID.COUNTRY_NAME, "BR"), # País
    x509.NameAttribute(x509.NameOID.STATE_OR_PROVINCE_NAME, "Distrito Federal"), # Estado
    x509.NameAttribute(x509.NameOID.LOCALITY_NAME, "Brasília"), # Cidade
    x509.NameAttribute(x509.NameOID.ORGANIZATION_NAME, "Universidade")
```

```

de de Brasília"), # Organização
    x509.NameAttribute(x509.NameOID.ORGANIZATIONAL_UNIT_NAME, "ST
I"), # Unidade Organizacional
    x509.NameAttribute(x509.NameOID.COMMON_NAME, "sti.unb.br"), # N
ome Comum (domínio)
])

```

# Constrói o certificado com as informações fornecidas

```

cert = (
    x509.CertificateBuilder()
        .subject_name(subject) # Define o nome do sujeito
        .issuer_name(issuer) # Define o nome do emissor (autoassinado, então
é o mesmo que o sujeito)
        .public_key(key.public_key()) # Define a chave pública
        .serial_number(x509.random_serial_number()) # Define um número de s
érie aleatório
        .not_valid_before(datetime.datetime.utcnow()) # Define a data de início
da validade
        .not_valid_after(datetime.datetime.utcnow() + datetime.timedelta(days=3
65)) # Define a data de término da validade (1 ano)
        .sign(key, hashes.SHA256()) # Assina o certificado com a chave privada
e o algoritmo SHA-256
)

```

# Salva a chave privada e o certificado em arquivos

```

for filename, data in [("key.pem", key.private_bytes(
    encoding=serialization.Encoding.PEM, # Codificação PEM
    format=serialization.PrivateFormat.TraditionalOpenSSL, # Formato tradic
ional do OpenSSL
    encryption_algorithm=serialization.NoEncryption())), # Sem criptografia
para a chave privada
    ("cert.pem", cert.public_bytes(serialization.Encoding.PEM))]: # Certifica
do em formato PEM
    with open(filename, "wb") as f:
        f.write(data) # Escreve os dados no arquivo

```

# Configura o servidor HTTPS

```

server_address = ('localhost', 4443) # Define o endereço do servidor (loca

```

```
lhost) e a porta (4443)
httpd = http.server.HTTPServer(server_address, http.server.SimpleHTTPRe
questHandler) # Cria o servidor HTTP

# Configura o contexto SSL
context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER) # Cria um context
o SSL para o servidor
context.load_cert_chain(certfile="cert.pem", keyfile="key.pem") # Carrega
o certificado e a chave privada

# Envolva o socket do servidor com o contexto SSL
httpd.socket = context.wrap_socket(httpd.socket, server_side=True)

# Inicia o servidor HTTPS
print("Servidor HTTPS rodando em https://localhost:4443")
httpd.serve_forever() # Mantém o servidor rodando indefinidamente
```

## Instalação da Biblioteca **cryptography**

Para instalar a biblioteca **cryptography** necessária para a geração de certificados SSL/TLS, execute o seguinte comando no terminal:

```
pip install cryptography
```

## Sobre a Biblioteca **cryptography**

A biblioteca **cryptography** é uma ferramenta robusta para a implementação de criptografia em Python. Ela oferece uma ampla gama de algoritmos criptográficos, incluindo RSA abordado anteriormente no seminário da disciplina, que é um dos métodos mais utilizados para a criptografia de chave pública. A **cryptography** é essencial para a geração de certificados SSL/TLS, que são fundamentais para estabelecer conexões seguras entre clientes e servidores.

## Geração dos Arquivos **cert.pem** e **key.pem**

No código fornecido, a biblioteca **cryptography** é utilizada para gerar uma chave privada RSA e um certificado autoassinado. A chave privada é salva no arquivo **key.pem**, enquanto o certificado é salvo no arquivo **cert.pem**. Esses arquivos são

usados pelo servidor HTTPS para criptografar a comunicação e autenticar sua identidade.

## Compilação e Execução do Código

Para rodar o servidor HTTPS, execute o seguinte comando no terminal dentro do diretório onde o script Python está salvo:

```
python trabalho2-Segurança.py
```

Isso iniciará o servidor e exibirá uma mensagem confirmando sua execução:

```
Servidor HTTPS rodando em https://localhost:4443
```

Agora o servidor está com conexões seguras na porta 4443.

## Acessando o Servidor pelo Navegador



Captura de tela 2025-02-17 221004

1. Abra um navegador e acesse: `https://localhost:4443`
2. Como o certificado é autoassinado, o navegador exibirá um aviso de segurança.
3. Clique em **Avançado** e prossiga para o site.

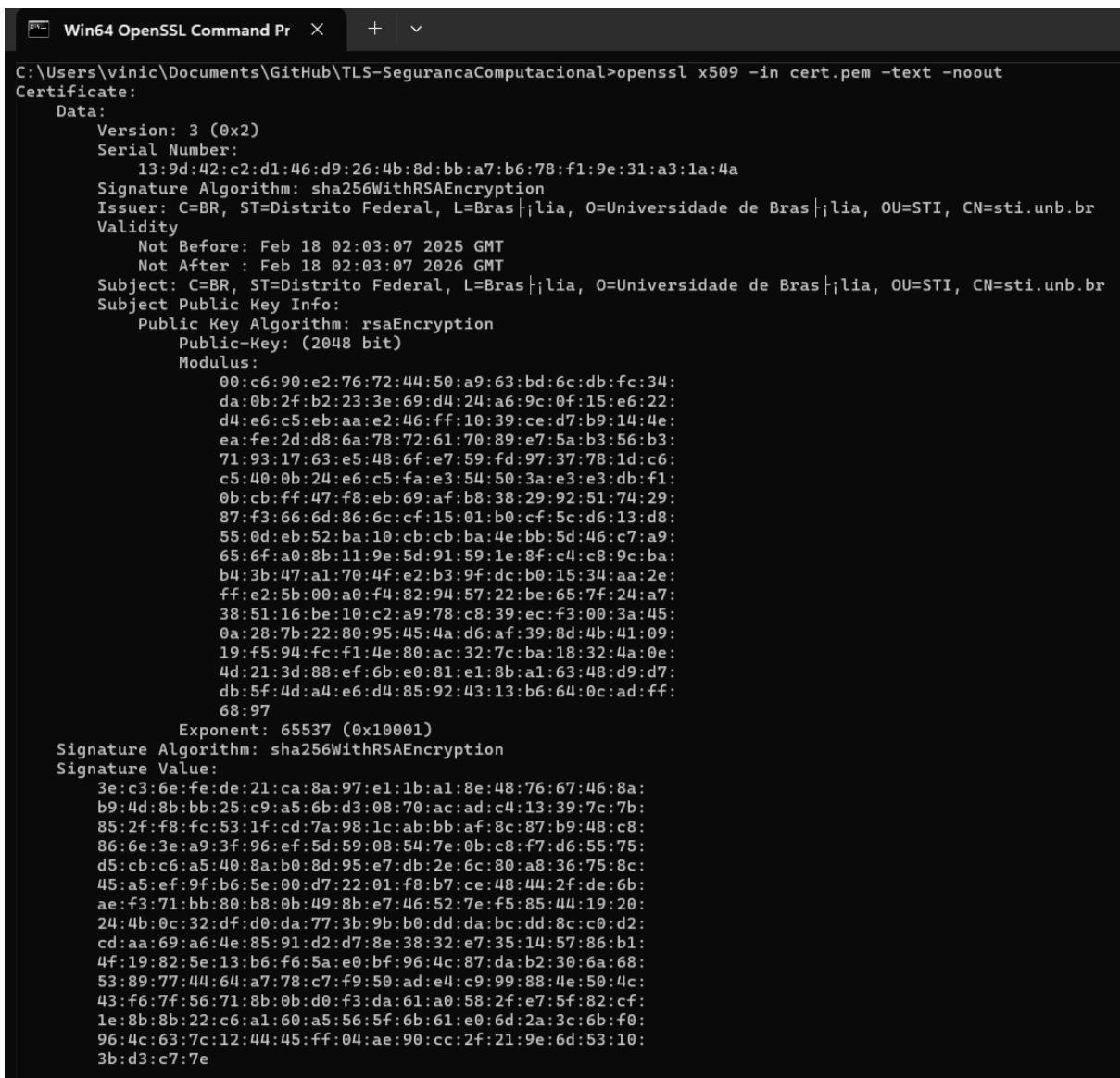
4. O servidor retornará a resposta HTTP segura configurada no código.

## Instalação do OpenSSL

Instale o OpenSSL no site [slproweb.com](http://slproweb.com) e baixe a versão “Win64 OpenSSL v3.4.1 Light”:

- Com o OpenSSL instalado, abra o terminal do programa (Win64 OpenSSL Command Prompt).
- Digite o comando abaixo, no diretório onde está implementado o servidor, para verificar a criação dos certificados na etapa anterior:

```
openssl x509 -in cert.pem -text -noout
```



```
Win64 OpenSSL Command Pr  X  +  v

C:\Users\vinic\Documents\GitHub\TLS-SegurancaComputacional>openssl x509 -in cert.pem -text -noout
Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      13:9d:42:c2:d1:46:d9:26:4b:8d:bb:a7:b6:78:f1:9e:31:a3:1a:4a
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=BR, ST=Distrito Federal, L=Brasília, O=Universidade de Brasília, OU=STI, CN=sti.unb.br
    Validity
      Not Before: Feb 18 02:03:07 2025 GMT
      Not After : Feb 18 02:03:07 2026 GMT
    Subject: C=BR, ST=Distrito Federal, L=Brasília, O=Universidade de Brasília, OU=STI, CN=sti.unb.br
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:c6:90:e2:76:72:44:50:a9:63:bd:6c:db:fc:34:
        da:0b:2f:b2:23:3e:69:d4:24:a6:9c:0f:15:e6:22:
        d4:e6:c5:eb:aa:e2:46:ff:10:39:ce:d7:b9:14:4e:
        ea:fe:2d:d8:6a:78:72:61:70:89:e7:5a:b3:56:b3:
        71:93:17:63:e5:48:6f:e7:59:fd:97:37:78:1d:c6:
        c5:40:0b:24:e6:c5:fa:e3:54:50:3a:e3:e3:db:f1:
        0b:cb:ff:47:f8:eb:69:af:b8:38:29:92:51:74:29:
        87:f3:66:6d:86:6c:cf:15:01:b0:cf:5c:d6:13:d8:
        55:0d:eb:52:ba:10:cb:cb:ba:4e:bb:5d:46:c7:a9:
        65:6f:a0:8b:11:9e:5d:91:59:1e:8f:c4:c8:9c:ba:
        b4:3b:47:a1:70:4f:e2:b3:9f:dc:b0:15:34:aa:2e:
        ff:e2:5b:00:a0:f4:82:94:57:22:be:65:7f:24:a7:
        38:51:16:be:10:c2:a9:78:c8:39:ec:f3:00:3a:45:
        0a:28:7b:22:80:95:45:4a:d6:af:39:8d:4b:41:09:
        19:f5:94:fc:f1:4e:80:ac:32:7c:ba:18:32:4a:0e:
        4d:21:3d:88:ef:6b:e0:81:e1:8b:a1:63:48:d9:d7:
        db:5f:4d:a4:e6:d4:85:92:43:13:b6:64:0c:ad:ff:
        68:97
      Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
    Signature Value:
      3e:c3:6e:fe:de:21:ca:8a:97:e1:1b:a1:8e:48:76:67:46:8a:
      b9:4d:8b:bb:25:c9:a5:6b:d3:08:70:ac:ad:c4:13:39:7c:7b:
      85:2f:f8:fc:53:1f:cd:7a:98:1c:ab:bb:af:8c:87:b9:48:c8:
      86:6e:3e:a9:3f:96:ef:5d:59:08:54:7e:0b:c8:f7:d6:55:75:
      d5:cb:c6:a5:40:8a:b0:8d:95:e7:db:2e:6c:80:a8:36:75:8c:
      45:a5:ef:9f:b6:5e:00:d7:22:01:f8:b7:ce:48:44:2f:de:6b:
      ae:f3:71:bb:80:b8:0b:49:8b:e7:46:52:7e:f5:85:44:19:20:
      24:4b:0c:32:df:d0:da:77:3b:9b:b0:dd:da:bc:dd:8c:c0:d2:
      cd:aa:69:a6:4e:85:91:d2:d7:8e:38:32:e7:35:14:57:86:b1:
      4f:19:82:5e:13:b6:f6:5a:e0:bf:96:4c:87:da:b2:30:6a:68:
      53:89:77:44:64:a7:78:c7:f9:50:ad:e4:c9:99:88:4e:50:4c:
      43:f6:7f:56:71:8b:0b:d0:f3:da:61:a0:58:2f:e7:5f:82:cf:
      1e:8b:8b:22:c6:a1:60:a5:56:5f:6b:61:e0:6d:2a:3c:6b:f0:
      96:4c:63:7c:12:44:45:ff:04:ae:90:cc:2f:21:9e:6d:53:10:
      3b:d3:c7:7e
```



## Wireshark: Analisando o Tráfego TLS

Para analisar o tráfego TLS utilizando o Wireshark, siga as instruções abaixo:

### Instalação do Wireshark

1. Acesse o site oficial do Wireshark:  
<https://www.wireshark.org/download.html>.
2. Baixe a versão apropriada para o seu sistema operacional (Windows, macOS, Linux).
3. Siga as instruções de instalação fornecidas pelo instalador.

### Capturando o Tráfego TLS na Porta 4443

#### Resumo da Análise de Tráfego HTTPS com Wireshark

##### 1. Identificação da Interface de Loopback

- Utilize o "Npcap Loopback Adapter" para capturar esse tráfego.

##### 2. Acessar o Servidor HTTPS

- Com a captura rodando, acesse no navegador: `https://localhost:4443`.
- O Wireshark começará a registrar os pacotes HTTPS.

##### 3. Aplicar Filtros para Focar no Tráfego TLS

- Filtrar todo o tráfego TLS: `tls`
- Filtrar pela porta 4443: `tcp.port == 4443`
- Para visualizar o handshake, busque pacotes como "Client Hello", "Server Hello" e "Certificate".
- Para acompanhar a comunicação completa, clique com o botão direito em um pacote → "Follow" → "TLS Stream".

##### 4. Observando o Handshake TLS

- O handshake define a versão do protocolo, algoritmos de criptografia e troca de certificados.
- No Wireshark, selecione um pacote "Client Hello" ou "Server Hello" e expanda:
  - **Transport Layer Security → Handshake Protocol**
- Verifique a versão do TLS (ex.: TLS 1.2 ou 1.3) e as cifras negociadas.

No.	Time	Source	Destination	Protocol	Length	Info
438	21.561750	127.0.0.1	127.0.0.1	TLSv1.3	285	Server Hello, Change Cipher Spec, Application ...
440	21.562390	127.0.0.1	127.0.0.1	TLSv1.3	74	Change Cipher Spec, Application Data
454	21.825749	127.0.0.1	127.0.0.1	TLSv1.3	1859	Client Hello (SNI=localhost)
456	21.828023	127.0.0.1	127.0.0.1	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application ...
458	21.828832	127.0.0.1	127.0.0.1	TLSv1.3	74	Change Cipher Spec, Application Data
468	21.869639	127.0.0.1	127.0.0.1	TLSv1.3	1795	Client Hello (SNI=localhost)
470	21.873260	127.0.0.1	127.0.0.1	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application ...
472	21.875132	127.0.0.1	127.0.0.1	TLSv1.3	124	Change Cipher Spec, Application Data
474	21.875566	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
475	21.875644	127.0.0.1	127.0.0.1	TLSv1.3	742	Application Data
477	21.875746	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
479	21.878562	127.0.0.1	127.0.0.1	TLSv1.3	270	Application Data
481	21.880323	127.0.0.1	127.0.0.1	TLSv1.3	13638	Application Data
490	22.133154	127.0.0.1	127.0.0.1	TLSv1.3	2098	Client Hello (SNI=localhost)
492	22.134255	127.0.0.1	127.0.0.1	TLSv1.3	285	Server Hello, Change Cipher Spec, Application ...
494	22.134891	127.0.0.1	127.0.0.1	TLSv1.3	124	Change Cipher Spec, Application Data
496	22.135268	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data

Frame 468: 1795 bytes on wire (14360 bits), 1795 bytes captured	0010 7f 00 00 01 7f 00 00 01 ca 53 11 5b 11 82 1c 3e	
Null/Loopback	0020 2a 87 54 09 50 18 00 ff 46 dc 00 00 16 03 01 06	*+T+P..
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0030 d2 01 00 06 ce 03 03 bc d2 62 92 c4 89 45 82 5a	
Transmission Control Protocol, Src Port: 51795, Dst Port: 4443,	0040 ee 4b 28 3a 8b 2f fe 4f 7c 40 9e 92 65 4b a6 66	K(:/-
Transport Layer Security	0050 16 a6 d7 e2 a4 fe 72 20 eb cc b5 62 bb a2 c5 37	.....r
TLSv1.3 Record Layer: Handshake Protocol: Client Hello	0060 e7 c3 a4 65 67 e1 73 6e 19 27 c7 ca 6c e5 24 22	.....eg s
Content Type: Handshake (22)	0070 77 f4 5c de 08 01 d5 c2 00 20 ca ca 13 01 13 02	w \ ....
Version: TLS 1.0 (0x0301)	0080 13 03 c0 2b c0 2f c0 2c c0 30 cc a9 cc a8 c0 13	...+/-
Length: 1746	0090 c0 14 00 9c 00 9d 00 2f 00 35 01 00 06 65 9a 9a	.....
Handshake Protocol: Client Hello	00a0 00 00 00 2d 00 02 01 01 00 00 00 0e 00 0c 00 00	.....
	00b0 09 6c 6f 63 61 6c 68 6f 73 74 00 17 00 00 00 2b	.....localh
	00c0 00 07 06 aa aa 03 04 03 03 00 0b 00 02 01 00 fe	.....
	00d0 0d 00 da 00 00 01 00 01 f1 00 20 a5 bd b3 4e 3d	.....
	00e0 ab 40 81 70 c6 1c af 17 5f 56 26 40 2f dc 48 a8	@ p ...
	00f0 af f8 6d e0 e9 e7 e2 27 ca ce 5a 00 b0 35 d9 ab	..m...
	0100 e4 ed 17 bb 71 b2 eb f9 a9 d1 8d d9 8c 98 ad 13	.....q...
	0110 bf 8d fa 98 d9 68 3d 9c 86 04 0b 3c 8d 33 23 1a	.....h=
	0120 8e b4 e9 e3 74 59 fd f8 27 0c 85 08 ea e1 a2 18	.....tY
	0130 2c f5 09 63 c2 92 4e 49 a2 b2 d6 8c e8 87 8e cb	,..c..

Captura de tela 2025-02-17 215148

## 1. Verificando os Dados Criptografados

- Após o handshake, os dados HTTP são criptografados.
- Pacotes de "Application Data" no Wireshark contêm apenas bytes aparentemente aleatórios.
- Não é possível visualizar requisições HTTP como GET ou POST em texto claro.
- Isso confirma que a comunicação está protegida por TLS e que terceiros não podem acessar os dados sem a chave de descryptografia.

No.	Time	Source	Destination	Protocol	Length	Info
551	52.915829	127.0.0.1	127.0.0.1	TLSv1.3	74	Change Cipher Spec, Application Data
557	52.918849	127.0.0.1	127.0.0.1	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application Data,...
559	52.919598	127.0.0.1	127.0.0.1	TLSv1.3	74	Change Cipher Spec, Application Data
570	53.168173	127.0.0.1	127.0.0.1	TLSv1.3	1795	Client Hello (SNI=localhost)
572	53.170555	127.0.0.1	127.0.0.1	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application Data,...
574	53.171098	127.0.0.1	127.0.0.1	TLSv1.3	74	Change Cipher Spec, Application Data
589	53.231773	127.0.0.1	127.0.0.1	TLSv1.3	1859	Client Hello (SNI=localhost)
591	53.233980	127.0.0.1	127.0.0.1	TLSv1.3	1490	Server Hello, Change Cipher Spec, Application Data,...
593	53.234675	127.0.0.1	127.0.0.1	TLSv1.3	124	Change Cipher Spec, Application Data
595	53.235093	127.0.0.1	127.0.0.1	TLSv1.3	858	Application Data
597	53.235169	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
599	53.235354	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data
601	53.441936	127.0.0.1	127.0.0.1	TLSv1.3	254	Application Data
603	53.449183	127.0.0.1	127.0.0.1	TLSv1.3	2409	Application Data
610	53.476477	127.0.0.1	127.0.0.1	TLSv1.3	2130	Client Hello (SNI=localhost)
616	53.554666	127.0.0.1	127.0.0.1	TLSv1.3	285	Server Hello, Change Cipher Spec, Application Data,...
618	53.555349	127.0.0.1	127.0.0.1	TLSv1.3	124	Change Cipher Spec, Application Data
620	53.555744	127.0.0.1	127.0.0.1	TLSv1.3	738	Application Data
621	53.555775	127.0.0.1	127.0.0.1	TLSv1.3	299	Application Data

Frame 595: 858 bytes on wire (6864 bits), 858 bytes captured (6864 bits) on interface 0	0020	19 c5 97 12 50 18 00 fa 3d 00 00 00 17 03 03 03	...
Null/Loopback	0030	29 76 75 9e 25 a7 96 40 2f f9 d9 da 6a ea 87 8b	vu·X·
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1	0040	7c 21 a9 ee 47 70 2f 21 7d 3b bb ff 86 01 db 7f	l·Gp/
Transmission Control Protocol, Src Port: 52177, Dst Port: 4443,	0050	8a 40 ef 8a 79 dc 83 88 0b ad a0 86 0e 97 de bc	@·y·
Transport Layer Security	0060	e2 a2 d3 2f 85 f6 a1 30 2c a6 78 68 4a f3 3e fa	··/·
TLSv1.3 Record Layer: Application Data Protocol: Application	0070	5d b7 48 2e 96 ac be d7 05 8b ab 4b a5 f6 00 70	]·H·
Opaque Type: Application Data (23)	0080	6a 1e a5 be 12 db 5a 7b 47 f6 a6 4d ef 77 d6 df	j·...·2
Version: TLS 1.2 (0x0303)	0090	6b 59 bb 8d 3e 7d a7 26 c9 a7 85 06 e6 5b 26 f4	kY·>}
Length: 809	00a0	31 e2 c9 1e f7 8a 09 84 0d 0c c4 f8 3a 57 4d af	1·...·
Encrypted Application Data [...]: 76759e25a796402ff9d9da6aea	00b0	9d 70 dd 9a e1 70 c8 22 cc 08 c9 cf a9 a3 c9 95	·p·p·
	00c0	44 5a 01 7f ae dc c4 59 ee a8 68 8d 91 93 ec f8	DZ·...·
	00d0	f5 b5 a4 70 c8 17 d9 65 42 b4 e4 f6 a0 d7 54 91	·p·p·
	00e0	ea a5 80 35 8b e2 fc 1c 85 04 a9 40 de 2f d4 46	·...5·
	00f0	72 a2 1c a3 2f 74 69 4e 15 b1 5c d0 ee ce 2e d6	r·.../ti
	0100	38 06 20 4a 2a ab 6a 88 d7 9c 4e 25 ea 14 41 60	8· J·j
	0110	9b 2b 2d 75 c2 df a1 18 96 ad 11 2e ad 68 0b 6d	+·u·
	0120	42 93 b6 17 18 f6 30 47 9c 68 cb 18 d2 15 1a 39	B·...·
	0130	84 23 3b 1c d1 97 68 65 94 c6 1f ee 41 4e 0c 85	·#·...·
	0140	f8 67 fb a4 79 96 21 8c 97 bc f0 ee f7 e4 ea 17	·g·y·
	0150	8e 3e 6e 01 cd 19 c9 48 ef d8 0e 4d ec 7a 94 97	·>n·
	0160	22 f0 36 92 18 30 69 bb ea 12 1c e4 f9 9d f5 db	·6·0·

Captura de tela 2025-02-17 220930

Análise dos Dados:

Ao clicar e expandir um desses pacotes, você perceberá que a seção de dados criptografados não apresenta os cabeçalhos HTTP (como GET, POST, etc.) e nem o corpo da mensagem em formato legível. Essa ausência de informações em texto claro indica que os dados foram criptografados conforme o esperado.

Conclusão:

O fato de os pacotes "Application Data" não exibirem conteúdo em texto plano é uma evidência de que a comunicação está protegida pelo TLS, garantindo que mesmo que alguém intercepte os pacotes, não poderá ler as informações sem a chave de descriptografia.

## 2.5. Preenchimento dos Campos do Certificado

**Geral**

Detalhes

**Emitido para**

Nome Comum (CN)	sti.unb.br
Organização (O)	Universidade de Brasília
Unidade Organizacional (UO)	STI

**Emitido por**

Nome Comum (CN)	sti.unb.br
Organização (O)	Universidade de Brasília
Unidade Organizacional (UO)	STI

**Período de Validade**

Emitido em	segunda-feira, 17 de fevereiro de 2025 às 22:57:15
Expira em	terça-feira, 17 de fevereiro de 2026 às 22:57:15

**Impressões digitais SHA-256**

Certificado	49dd9e37ff9fa4c3d3674ec905702bfe7729065708473aedc99f24e67e99d437
Chave Pública	0a110650db76cfb4daac8ee1749df2452b18350878a7c37d564d51fbf797e5ca

Geral

**Detalhes**

## Hierarquia de Certificados

sti.unb.br

## Campos de Certificado

## ▼ Informações da Chave Pública do Assunto

Algoritmo de Chave Pública do Assunto

Chave Pública da Entidade

Algoritmo de Assinatura de Certificado

Valor da Assinatura do Certificado

## ▼ Impressões digitais SHA-256

Certificado

Chave Pública

## Valor do Campo

Módulo (2048 bits):

C6 90 E2 76 72 44 50 A9 63 BD 6C DB FC 34 DA 0B  
2F B2 23 3E 69 D4 24 A6 9C 0F 15 E6 22 D4 E6 C5  
EB AA E2 46 FF 10 39 CE D7 B9 14 4E EA FE 2D D8  
6A 78 72 61 70 89 E7 5A B3 56 B3 71 93 17 63 E5

Exportar...

Visualizador de Certificados: sti.unb.br

Geral

**Detalhes**

Hierarquia de Certificados

sti.unb.br

Campos de Certificado

▼ sti.unb.br

▼ Certificado

Versão

Número de Série

Algoritmo de Assinatura de Certificado

Emissor

▼ Validade

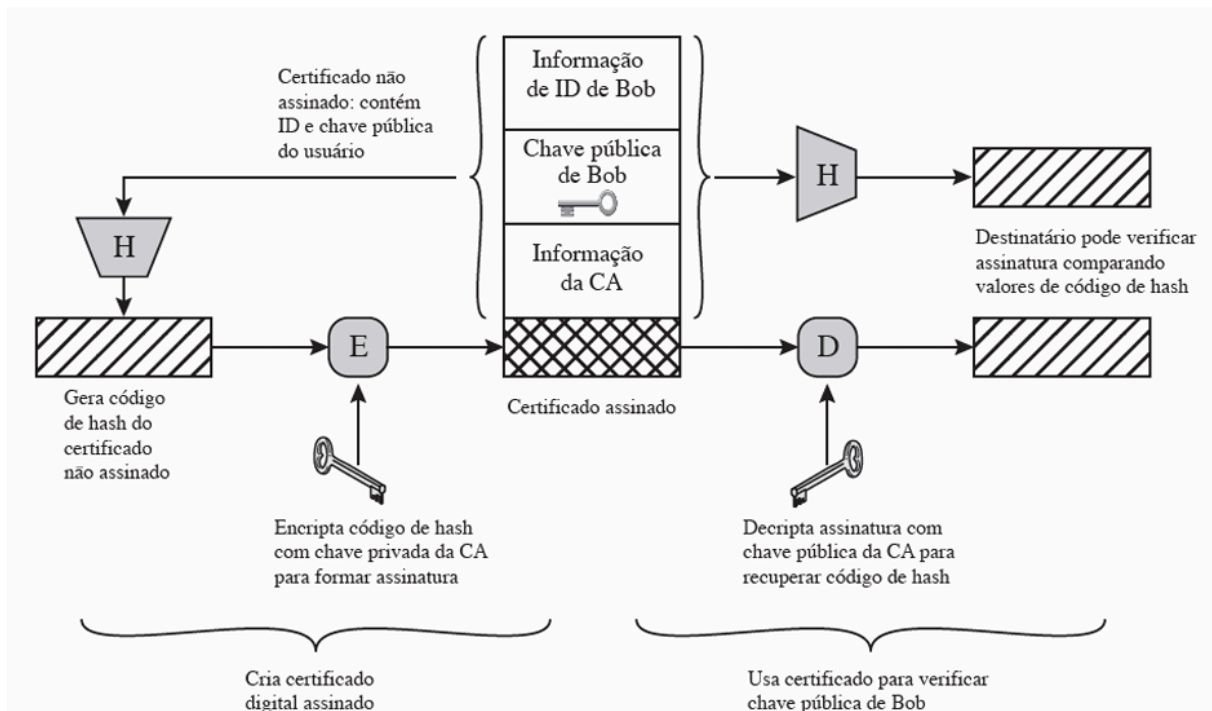
Não Antes

Valor do Campo

PKCS #1 SHA-256 Com Criptografia RSA

Exportar...

Ao criar um certificado digital, diversos campos podem ser preenchidos para garantir a correta identificação da entidade proprietária do certificado. Esses campos são essenciais para assegurar autenticidade e confiança na comunicação segura. Abaixo, explicamos os principais campos e sua importância:

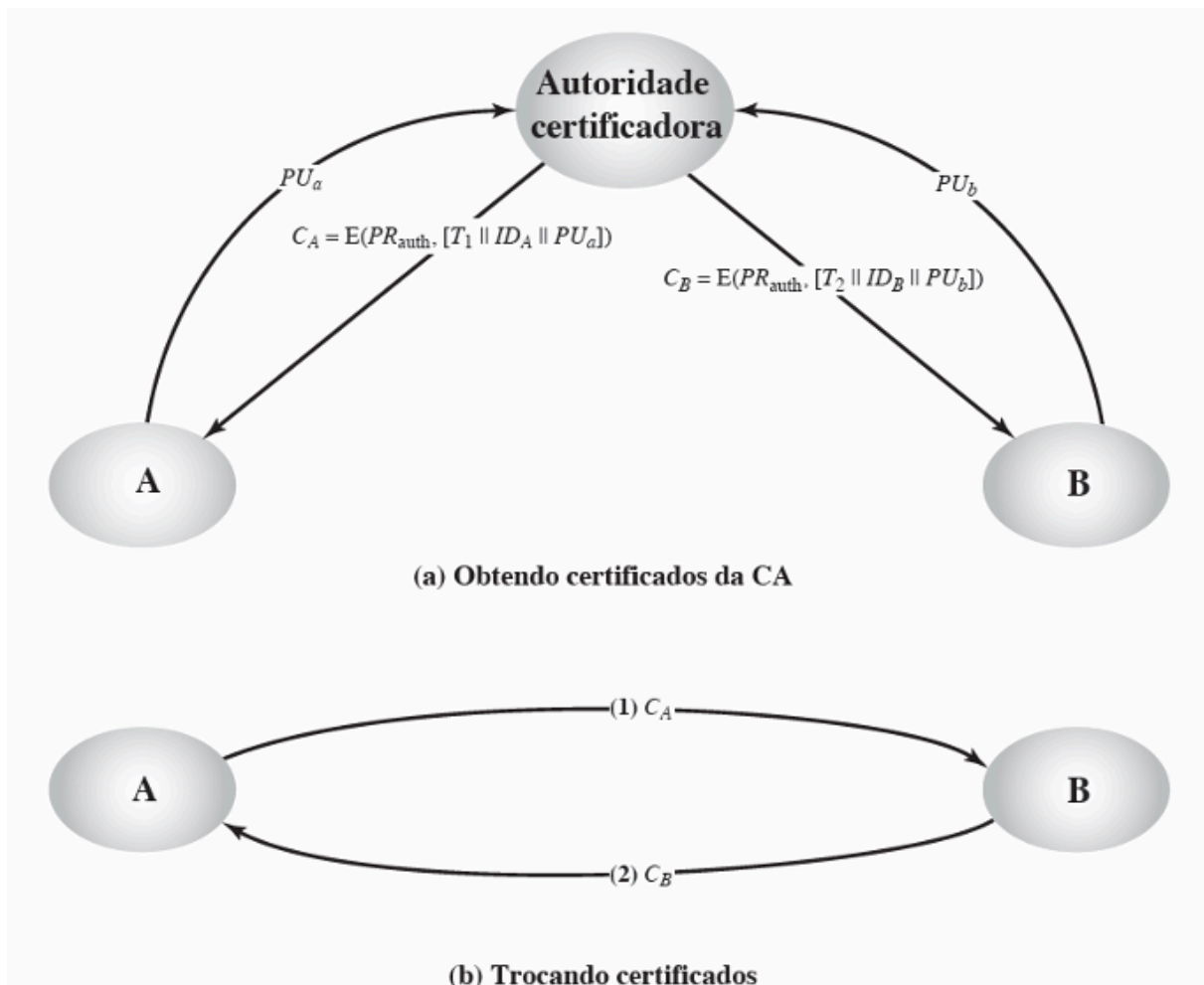


## Protocolo X.509

O protocolo X.509 é um padrão para a infraestrutura de chave pública (PKI) utilizado na emissão de certificados digitais. Esses certificados são essenciais para a implementação do TLS (Transport Layer Security) na web, garantindo a segurança das comunicações entre clientes e servidores. Um certificado X.509 contém informações como a chave pública do servidor, a identidade do proprietário (nome comum, organização, país, etc.), o período de validade e a assinatura digital de uma Autoridade Certificadora (CA) confiável. Quando um navegador se conecta a um site HTTPS, ele verifica o certificado X.509 apresentado pelo servidor para assegurar que a conexão é segura e que o servidor é autêntico. O uso de certificados X.509 é fundamental para estabelecer conexões seguras e proteger os dados transmitidos na web.

- **C = Country (País)** → Indica o país onde a organização ou entidade está registrada. Exemplo: **BR** para Brasil.
- **ST = State/Province (Estado ou Província)** → Especifica o estado ou província dentro do país. Exemplo: **Distrito Federal**.
- **L = Locality (Cidade ou Localidade)** → Define a cidade onde a entidade está localizada. Exemplo: **Brasília**.
- **O = Organization (Organização)** → Nome da empresa ou entidade proprietária do certificado. Exemplo: **Universidade de Brasília**.

- **OU = Organizational Unit (Unidade Organizacional)** → Usado para definir um departamento dentro da organização. Exemplo: `TI`.
- **CN = Common Name (Nome Comum)** → Define o domínio ou nome da entidade que usará o certificado. Exemplo: `localhost` ou `sti.unb.br`.



Captura de tela 2025-02-17 221726

Como este é um trabalho da disciplina de segurança computacional, o certificado gerado não é válido para HTTPS em um ambiente real. Para que um certificado seja considerado válido, ele deve ser emitido por uma Autoridade Certificadora (CA) confiável, como a ICP-Brasil no Brasil ou a DigiCert internacionalmente. Cada navegador vem de fábrica com uma lista de autoridades certificadoras confiáveis e verifica a autenticidade do certificado através do protocolo TLS. Isso garante que o site acessado pelo usuário é certificado e que o tráfego é criptografado utilizando TLS e RSA, assegurando a confidencialidade e integridade dos dados transmitidos.



Os certificados TLS são fundamentais para a segurança na web. Eles são usados para estabelecer conexões seguras entre clientes e servidores, garantindo que os dados transmitidos sejam criptografados e protegidos contra interceptações. Um certificado TLS contém informações sobre a identidade do servidor, a chave pública usada para criptografia e a assinatura digital da CA que emitiu o certificado. Quando um navegador se conecta a um site HTTPS, ele verifica o certificado apresentado pelo servidor contra a lista de CAs confiáveis. Se o certificado for válido, a conexão segura é estabelecida, permitindo a troca de dados de forma segura e protegida.

---

### **3. Conclusão e Análise Final**

A implementação do HTTPS melhora significativamente a segurança das aplicações web, garantindo a confidencialidade, integridade e autenticidade dos dados. Apesar dos desafios na configuração e implementação, o uso de certificados autoassinados permitiu um entendimento mais aprofundado sobre a criptografia na web. Futuramente, melhorias como a integração com autoridades certificadoras podem ser exploradas para aumentar a confiabilidade da solução.

---

### **4. Bibliografia**

Stallings, William. Cryptography and Network Security: Principles and Practice, Global Edition. Germany, Pearson Education, 2022.